Context Dependent Revocation in Delegated XACML

Erik Rissanen[1], Ludwig Seitz[2]

[1] Axiomatics AB
Electrum 223, 164 40 Kista, Sweden
erik@axiomatics.com

[2] Swedish Institute of Computer Science, SICS AB
Box 1263, Kista 16429, Sweden
ludwig@sics.se

June 2008

SICS Technical Report T2008:10
ISSN 1100-3154

Abstract

The XACML standard defines an XML based language for defining access control policies and a related processing model. Recent work aims to add delegation to XACML in order to express the right to administrate XACML policies within XACML itself. The delegation profile draft explains how to validate the right to issue a policy, but there are no provisions for removing a policy. This paper proposes a revocation model for delegated XACML. A novel feature of this model is that whether a revocation is valid or not, depends not only on who issued the revocation, but also on the context in which an attempt to use the revoked policy is done.

Keywords: XACML, Access Control, Revocation

1 Introduction

XACML [10], eXtensible Access Control Markup Language, is an XML based standard language for defining access control policies. Recently the XACML technical committee has been working on adding delegation to XACML in order to express the right to administrate XACML policies within XACML itself. The work is at the time of writing still in progress, but there is a fairly developed draft available, [7]. The delegation profile draft explains how to validate the right to issue a policy, but there is no support to validate the right to remove, or revoke, a policy. This paper proposes a revocation model for delegated XACML.

The properties of delegated XACML lead to a model in which the validity of a revocation depends, not only on who issued the revocation, but also on the context in which the policy that was revoked is used. For instance, a single policy issued by a single issuer could grant the right to use both a printer and a web page. This single policy could in delegated XACML be supported by two different authorities: a different authority for the printer than for the web page. In case a printer authority revokes the joint policy, then revocation would have effect only when the policy is used to access the printer, not when it is used to access the web page.

2 Introduction to XACML

Due to space constraints we can only give a brief overview of XACML here. The XACML committee website [10] contains a good introduction to the basic principles of XACML [9], the specification of the XACML language and processing model [6] and a draft for the upcoming delegation model [7].

XACML is based on attributes of subjects, resources, actions and the environment, which are used to describe an access *request*. The box labelled “Request 1” in figure 1 is an illustration of a request. In this case the request asks “Is the access allowed where the subject is Bob and the resource is Printer 14?”.

Additional attributes to those in the request can be provided by external sources. The complete set of attributes used in the processing of a request is known as the *request context*. There is a table of such additional attributes in figure 1.

Note that in this paper we do not consider the source of attributes or how attributes are administered. We simply assume that the attributes are available and are trusted. *Policies* can then be used to express rights in terms of the attributes.

XACML policies are functional expressions that pull their input from the request context. When a request context is evaluated against a set of policies, a response is calculated as specified by the semantics of the policy format and the processing model. The response can either be *Permit*, *Deny*, *NotApplicable* or *Indeterminate*.

2.1 Delegation

The right to issue policies is sometimes called delegation. The word delegation has various meanings in the access control literature. For our purposes it denotes the act of creating rights.

Delegated XACML differentiates between trusted policies and policies with untrusted issuers. The trusted policies need no special validation and form the roots of trust (analogous to the self signed certificates of root CAs in PKI). The policies with untrusted issuers have to be validated against the trusted policies. The validation process is called *reduction* and is performed when an access request is evaluated.

Reduction is performed by means of a search in a directed graph. The nodes of the graph are the policies in the policy set where reduction is performed. The edges of the graph represent that one policy is authorised by another and are calculated with *administrative requests* which contain the attributes of the issuers of policies. The issuer attributes are contained in a category called *Delegate*. Figure 2 shows an example of an administrative request. The meaning of the administrative request is "May John issue a policy which grants Bob access to Printer14?" Such a request can establish an edge between a policy issued by John, and a policy which grants the authority. Policies 1 to 4 in figure 1 are examples of such *administrative policies* which grant the right to issue other policies.

Delegated XACML introduces an issue concerning the timing of attributes. When the right to issue a policy is validated, the attributes of the issuer are compared against conditions in the administrative policies. Some attributes may change over time, so the question is for which point in time the attributes of an issuer should be resolved. The XACML delegation draft mentions two possibilities: use the attributes of issuers as they are at the time of the access request or as they were at the time the policy being reduced was issued.

In this paper the second option is used since in this case a policy remains valid when an administrator leaves and loses his administrator attributes, which we find desirable.

An example of a reduction graph, can be seen in figure 1.

3 Revocation

The XACML administrative policy draft [7] does not in any way address the issue of how policies can be revoked.

Please note that we do not address distribution of revocations in this paper. For the sake of this paper, we assume that the revocations have been distributed and are available. Our work is then about who may revoke which policy, which becomes more complex in the case of delegated XACML, than for instance in the case of Public Key Infrastructure, PKI.

We define a revocation as a statement made by an issuer which contains:

1. An identifying attribute of the issuer.

2. The identifier of the policy which is being revoked.

The semantics of the revocation is that the issuer asserts that the revoked policy must not be in effect. Our goal here is to define the authority of a given issuer of a revocation with respect to a given revoked policy.

We base our revocation model on a use case where there is an administrator group in an organisation. The tasks of the administrators include providing access control policies for resources. The administrator group is expressed as an attribute. The same group should also be able to remove the policies they can create.

Using the delegation model we can allow a some group to issue policies within specified constraints. For instance, policy 1 in figures 1 and 4 implies that everyone in the Printer_Admins group has the right to issue policies about the resources in the group Printers, since policy 1 will support policies which match requests on those resources.

For our revocation model we have the following requirements based on our use case:

1. A current administrator can revoke policies which he could issue himself, regardless of whether it was himself who issued the policy.

2. A former administrator cannot revoke a policy, even if it was he himself who issued it.

3. (Optionally) a current administrator should also be able to revoke policies that are indirectly supported by policies he could issue.

Note that the simple and common revocation model where the issuer of a policy (or a certificate) may revoke it, is not appropriate for these requirements.

Also note that we do not want to let the issuer of the policy to decide who has authority to revoke. For instance, embedding an URL of a revocation list in the policy is not acceptable (see for instance section 4.2.1.14 in [5]), since that would leave revocation in the control of the issuer.

As we will show, the third requirement makes the reduction process NP-complete, so this requirement would likely be sacrificed in an implementation. In the interest of research, we keep it in this paper and the examples shown.

3.1 Model description

We now present our revocation model. The model consists of a modification to the processing of delegation in XACML. Initially, the reduction graph is formed as specified in the XACML administrative policy draft [7]. However, the search of the graph is modified by our revocation model.

Consider the search process, during which we are about to explore an edge from policy P_a to policy P_b. (For simplicity, we ignore the different types of edges present in the full reduction model.)

Let $Path_{P_a}$ denote the reduction path that was used to reach P_a. To form the edge from P_a to P_b, an administrative request, $AReq$, was used.

1. For each revocation R_j of a policy on the reduction path path including P_a, $Path_{P_a} \cup \{P_a\}$, generate a *Revocation Authorisation Request*, RAR_j. The revocation authorisation requests are used to validate the authority of the revocations. A RAR has the same form as an XACML administrative request and its content is based on $AReq$.

2. RAR_j is generated from the edge generating request $AReq$ by replacing the Delegate category of $AReq$ with the issuer of the revocation R_j.

3. Evaluate each RAR_j against the policy P_b.

4. If any RAR_j evaluates to Permit, then the search does not make the jump from policy P_a to policy P_b.

Informally, we stop the graph search if we find an administrative policy which supports a revocation of a policy on the current reduction path during the search, since the revocation breaks the path. Note that we stop the search before we reach a trusted policy. This means that the administrative policy P_b that authorises the revocation R_j might be unauthorised. However, this is not an issue since the end result is the same regardless the policy P_b is authorised or not: in either case P_a is not authorised through P_b. There are two possibilities.

If P_b is not authorised, we would not be able to reduce to the trusted issuer through P_b, meaning P_a would not be authorised through P_b even though the revocation R_j would be unauthorised.

If the policy P_b is authorised, then the revocation R_j is also authorised and P_a is not authorised through P_b because of the revocation on the path.

3.2 Example

Figure 1 shows an example.

Consider when policy 5 is reduced by means of a search of the reduction graph, and we have reached policy 3, and are about to jump the edge to policy 1. The reduction path consists of policy 5. The edge between policy 3 and policy 1, was generated by an administrative request, $AReq$. In this case $AReq$ was the one shown in Figure 2. (In terms of the model description above, we have P_a = Policy 3 and P_b = Policy 1.)

1. In the example revocation 1 is a revocation on policy 5, which is on the reduction path. We need to test whether this revocation is valid. We do so by generating the Revocation Authorisation Request, RAR.

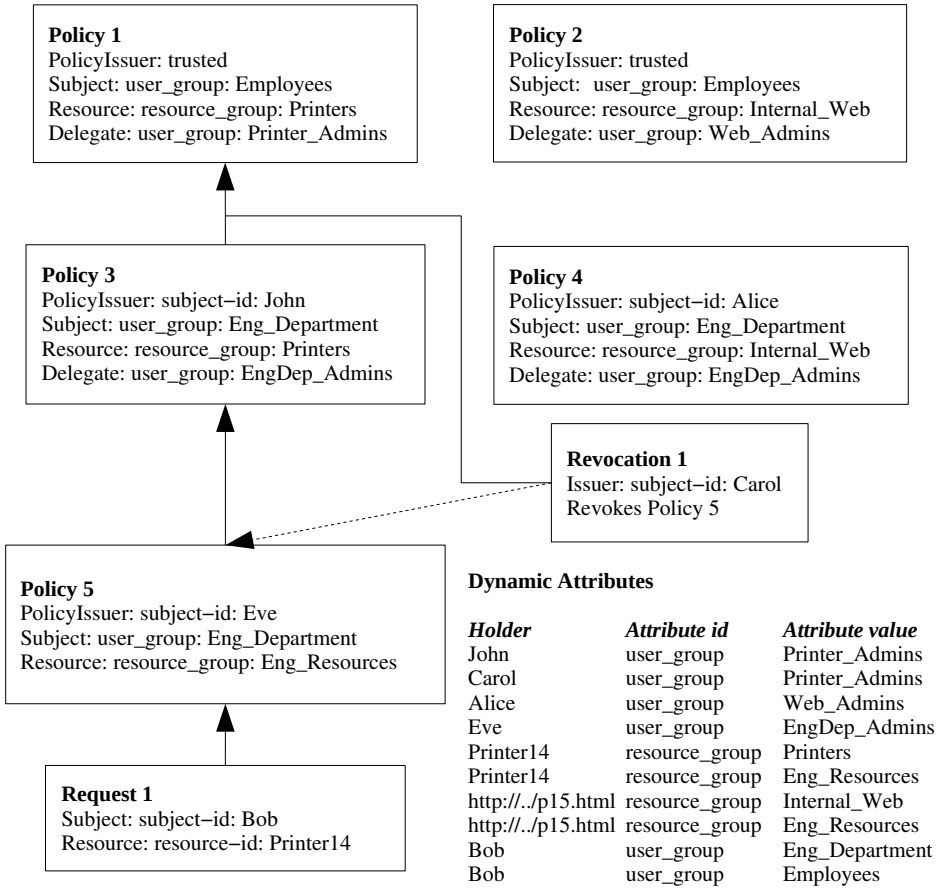

Figure 1: Dependencies between policies and the revocation. Policy 5 combines resources from different authorities so its revocation status depends on who revoked it. Revocation 1 is issued by a printer administrator, so when the access request is about access to a printer, revocation 1 gets support from the same path as the access request, invalidating the path to the trusted policy. The request is not permitted.

2. The RAR is generated from the administrative request $AReq$ and the revocation by replacing the Delegate of $AReq$ with the issuer of the revocation. In this case the result is the RAR shown in Figure 3.

3. We evaluate the RAR against policy 1. In this case, since Carol is a member of the Printer_Admins group, the RAR will evaluate to Permit.

4. Since the RAR evaluates to Permit with policy 1, the search is stopped here and we do not follow the edge from policy 3 to policy 1.

5. In this case there is no other path to backtrack to, so policy 5 will not be authorised.

```
<Request>
  <Attributes Category="&Delegate;">
    <Attribute AttributeId="&subject-id;">
      <AttributeValue DataType="&string;">John</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="&Subject;">
    <Attribute AttributeId="&subject-id;">
      <AttributeValue DataType="&string;">Bob</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="&Resource;">
    <Attribute AttributeId="&resource-id;">
      <AttributeValue DataType="&string;">Printer14</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```

Figure 2: The administrative request which is used to reduce policy 3.

In delegated XACML it is possible that a delegated policy receives support from different policies depending on the access situation. This reflects that resources with different authorities have been composed into a single policy. This possibility is handled by our revocation model.

For instance, if we instead have the request in figure 4, the revocation will not have any effect. First, policy 5 will match Request 2 since the web page is in the EngDep_resources group. For this request the resource does not match the conditions in policies 1 and 3, so the search does not reach them. Instead policy 4 will support policy 5 and policy 2 will in turn support policy 4. Revocation 1 has no effect on this path since it is not supported by either policy 4 or by policy 2.

This is as desired since Carol, being a printer administrator should not have any authority over web pages. With this revocation model, a policy which combines resources from different authorities may be either revoked or not revoked depending on which authority revoked the policy and which resource the access request concerns. If a printer administrator revoked the policy, it is revoked in the context of an access request to a printer, but not in the context of a web page.

3.3 Computational complexity

If *indirect revocation*, that is revocation along the full reduction path is allowed, the revocation model makes the reduction search of XACML delegation NP-complete.

If indirect revocation is not allowed, it is not necessary to search the full reduction path for revoked policies, so the graph search is no longer path dependent and the search is not NP-complete and the complexity of the XACML delegation model is not affected.

Here we present an informal sketch of a proof showing that indirect revocation

```
<Request>
  <Attributes Category="&Delegate;">
    <Attribute AttributeId="&subject-id;">
      <AttributeValue DataType="&string;">Carol</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="&Subject;">
    <Attribute AttributeId="&subject-id;">
      <AttributeValue DataType="&string;">Bob</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="&Resource;">
    <Attribute AttributeId="&resource-id;">
      <AttributeValue DataType="&string;">Printer14</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```

Figure 3: The revocation authorisation request which is used to check the authority of Carol to revoke policy 5 when policy 3 is being reduced.

makes the reduction search NP-complete. We reduce to 3-SAT, which is known to be NP-complete.

Consider the following instance of 3-SAT:

$$\begin{gathered}(x_{1,1}\ OR\ x_{1,2}\ OR\ x_{1,3})\ AND \\ (x_{2,1}\ OR\ x_{2,2}\ OR\ x_{2,3})\ AND \\ \ldots \\ (x_{N,1}\ OR\ x_{N,2}\ OR\ x_{N,3})\end{gathered}$$

where each x is a variable or a negation of a variable.

Based on this 3-SAT instance we can generate an XACML policy set, which together with indirect revocations will solve the 3-SAT instance.

We create a policy set which contains the following policies:

- For $x_{1,1}$, $x_{1,2}$ and $x_{1,3}$, we create policies $P_{1,1}$, $P_{1,2}$ and $P_{1,3}$. The policies have unique issuers $I_{1,1}$, $I_{1,2}$ and $I_{1,3}$. These policies are access policies for a dummy resource called $Resource$.
- For the remaining x, we create policies which allow delegation to the issuers of the policies corresponding to the preceding disjunction in the 3-SAT instance. That is, policies $x_{k,1}$, $x_{k,2}$ and $x_{k,3}$ have issuers $I_{k,1}$, $I_{k,2}$ and $I_{k,3}$. Each of these policies allow a disjunction of three delegates $I_{k-1,1}$, $I_{k-1,2}$ or $I_{k-1,3}$. Again, the resource allowed by the policies is the dummy resource.
- There is one root policy which allows a disjunction of the three delegates $I_{N,1}$, $I_{N,2}$ or $I_{N,3}$ (and the dummy resource).

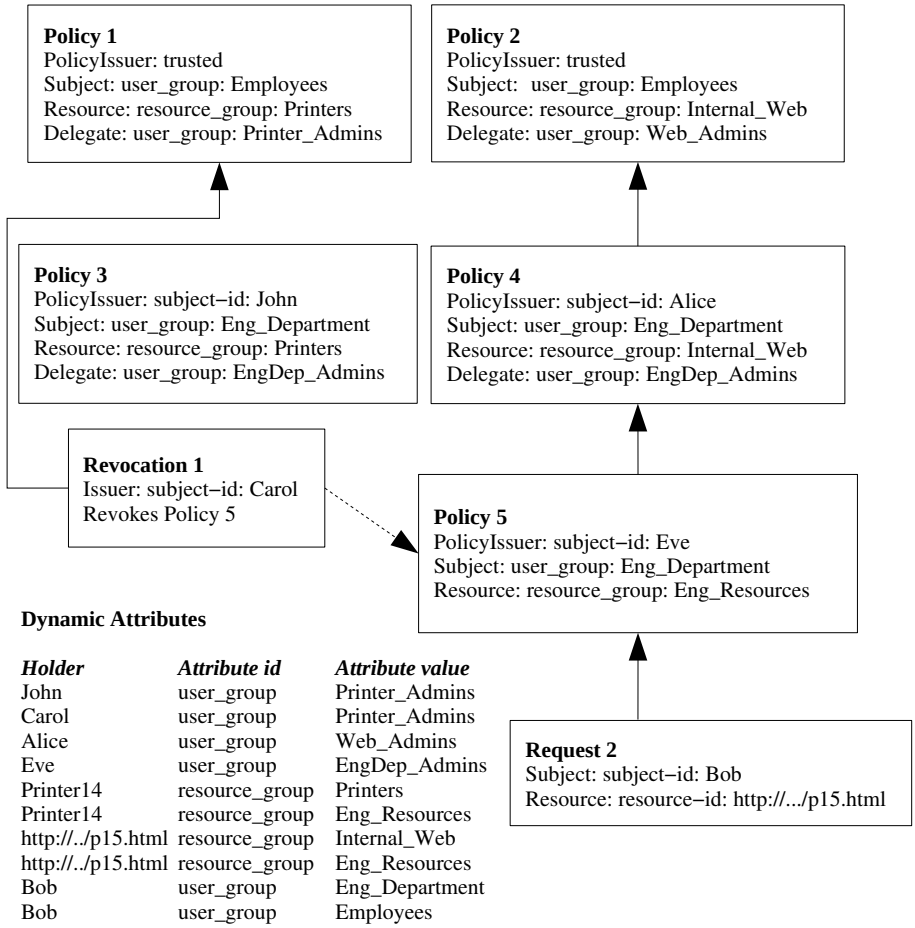

Figure 4: Dependencies between policies and the revocation. Revocation 1 is issued by a printer administrator, so when the access request is for a web page, the revocation does not get any support from the path that supports the access request. The revocation has no effect in the context of this access request and the access is permitted.

See figure 5 for an illustration of the resulting reduction graph.

In addition to the policies, we create revocations. For each pair $(x_{i,j}, x_{k,l})$, where $i < k$ and $x_{i,j}$ and $x_{k,l}$ cannot be satisfied simultaneously, we create a revocation for policy $P_{i,j}$ issued by issuer $I_{k,l}$.

When we evaluate an access request to the dummy resource in this policy set, the 3-SAT instance is satisfiable iff we can find a reduction path to the trusted root policy.

First, assume that the 3-SAT instance is satisfiable. In this case we can select a path corresponding to the variable assignment which satisfies the 3-SAT. This path will not contain any revocations since we create revocations only if there is a

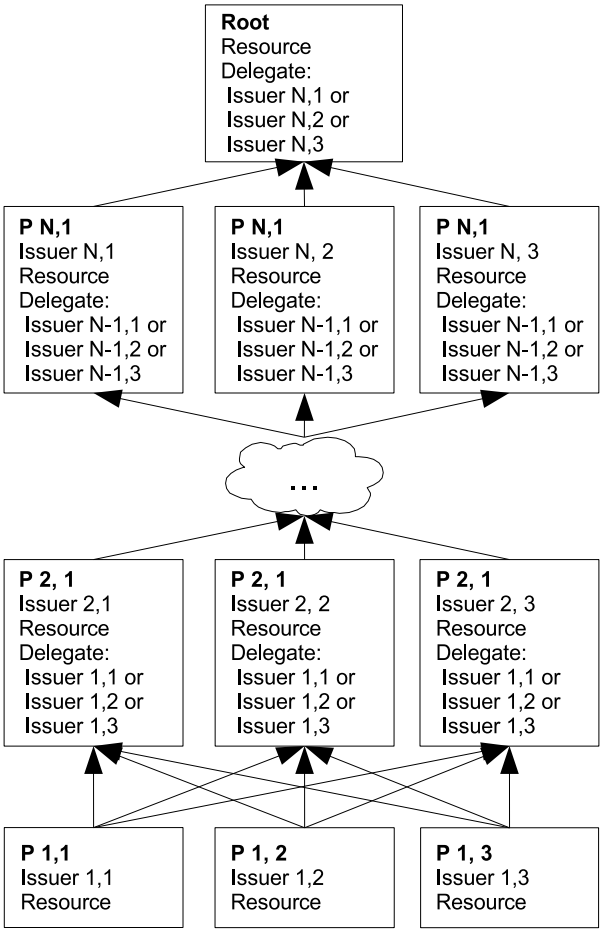

Figure 5: The reduction graph in the policy set used to solve the 3-SAT instance. The structure corresponds to the clauses in the 3-SAT instance. The interpretation is such that if we find a reduction path to the trusted root policy, we have satisfied the 3-SAT in the way we selected the policies on the reduction path. Revocations (not shown) are used to block paths which correspond to conflicts in the 3-SAT assignment.

conflict in the variable assignments.

Second, assume that there is a valid, non-revoked reduction path in the policy set. In this case we can satisfy the 3-SAT by setting the corresponding x to true. There cannot be any conflict in the assignment since we assumed that there was no revocation on the path.

It is easily seen that the time needed to construct the policy set and the revocations is polynomial, so indirect revocation reduces to 3-SAT and is NP-complete.

3.4 Properties of the model and related work

Informally, in our model someone with the authority to support a policy, either directly or indirectly, can also revoke it. The motivation for choosing this model is that in many cases it is natural for the right to issue and revoke to go together. Also, being able to indirectly support a policy implies authority over the policy, so it is not far fetched to allow for revocation of indirectly supported policies. An alternative would be to only allow to revoke policies which are directly supported, which has the benefit that such a model is not NP-complete. Note however, that since our revocations cascade, revoking a directly supported policy also invalidates indirectly supported policies.

Hagström et.al. [4] classify revocation schemes based on three characteristics: *resilience*, *propagation* and *dominance*.

Resilience refers to whether a permission stays revoked if it would be reissued. In other words, resilience differentiates between negative permissions which override a positive permission and removal of a permission. In our model revocations are not resilient since they refer to specific instances of policies.

Propagation refers to whether revocation of a permission which has been used for delegation will cause a cascading removal of dependent permissions. Our model is cascading since a revocation blocks the reduction process, thus invalidating everything that was supported by the revoked policy. Other models allow for both cascading and noncascading revocation, such as those explored database research [2] and in RBAC [8]. Note that the effect of noncascading revocation can be achieved in our model by not revoking a policy, but instead revoking an attribute.

Finally, *dominance* refers to what happens to a permission if it is revoked but the permission is also granted by other grantors. The revocation is said to be dominating, or strong, if the permission goes away despite this other support. Otherwise it is known as weak. Note that Hagström et.al. only consider the cases where the other grant is not independent of the one being revoked, that is there is a common supporting permission for both grants. Our model is weak since if there are two policies granting the same permission, a revocation will affect only either instance, leaving the other.

E. Barka and R. Sandhu [1] introduce another characteristics which they call *grant-dependency*. In what they call a *grant-independent* delegation, any member of the delegator role can revoke the membership of a delegatee. If one generalises this to an attribute based approach such as used in XACML, this corresponds to our *you can revoke what you could grant* approach. Our model is *grant-independent* since not only the original issuer can revoke a policy.

Sadighi et.al. [3] present a feature rich revocation model for their Privilege Calculus delegation model, which is somewhat similar to the XACML delegation model. In contrast to the XACML delegation model, the privilege calculus takes time into account in administrative actions, which permits both cascading and noncascading revocations by setting different timestamps for when a revocation should take effect.

The central novel feature of our model, which is not captured by any of the characteristics described above, is that the effect of a revocation depends on which access request is being processed. We call this characteristics *context dependency*. A revocation which is context independent will have the same effect on a permission regardless for what the permission is used. We are not aware of any other access control model which has context dependent revocation.

4 Future work

One issue with our model is that a revocation does not remove a policy, rather a revocation means more information to be considered during access request processing. This leads to two concerns.

Firstly, there is the concern that the database with access control information grows over time. However, in a setting with digital signatures policies would likely expire regardless the revocation model, but the issue warrants further research.

The second concern is that context dependent revocation could be hard for human administrators to comprehend. We plan to explore this issue in the context of our ongoing research on support tools for policy administration.

References

[1] BARKA, E., AND SANDHU, R. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference* (New Orleans, Louisiana, USA, December 2000).

[2] BERTINO, E., SAMARATI, P., AND JAJODIA, S. An Extended Authorization Model for Relational Databases. *Transactions on Knowledge and Data Engineering 9*, 1 (1997), 1–17.

[3] FIROZABADI, B. S., AND SERGOT, M. Revocation Schemes for Delegated Authorities. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks* (June 2002).

[4] HAGSTRÖM, Å., JAJODIA, S., PARISI-PRESICCE, F., AND WIJESEKERA, D. Revocations - a Classification. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations* (Cape Breton, Nova Scotia, Canada, June 2001).

[5] HOUSLEY, R., POLK, W., FORD, W., AND SOLO, D. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Request For Comments (RFC) 3280, Internet Engineering Task Force (IETF), April 2002.
http://www.ietf.org/rfc/rfc3280.txt.

[6] MOSES, T. eXtensible Access Control Markup Language (XACML) Version 2.0. Standard, Organization for the Advancement of Structured Information Standards (OASIS), February 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[7] RISSANEN, E., LOCKHART, H., AND MOSES, T. XACML v3.0 administrative policy. Standard, Organization for the Advancement of Structured Information Standards (OASIS), May 2007. http://www.oasis-open.org/committees/download.php/23951/xacml-3.0-admininstration-v1-wd-17.zip.

[8] SANDHU, R., BHAMIDIPATI, V., AND MUNAWER, Q. The ARBAC97 Model for Role-Based Administration of Roles. *Transactions on Information and System Security (TISSEC) 2*, 1 (1999), 105–135.

[9] XACML, 2003. http://www.oasis-open.org/committees/download.php /2713/Brief_Introduction_to_XACML.html.

[10] XACML, 2004. http://www.oasis-open.org/committees/xacml.